VSIPL++/FPGA Design Methodology

Jules Bergmann, Air Force Research Laboratory/IFTC
Susan Emeny, ITT
Peter Bronowicz, ITT

We describe a hardware/software codesign methodology for hybrid hardware and software systems. The methodology integrates VSIPL++ for software design and a portable, composable hardware design method based on streams. The hardware design is portable and scalable from design/test systems to the target system and to future technologies. The methodology increases productivity by providing a concise function description in both hardware and software and by providing a streamlined interface between hardware and software. The methodology supports a design methodology from algorithms to embedded systems with hardware/software co-design, strong unit and system testing, and virtual breadboarding. It simplifies the integration of hardware and software to create high performance applications. It enables the use of predefined FPGA libraries for application acceleration.

A standard high-level synthesis hardware design methodology, using a register Transfer Logic (RTL) description in a Hardware Design Language (HDL), achieves portability and scalability. The design can be synthesized onto a range of Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuit (ASICs). Encapsulation of device specific optimizations into macro cells ensures high performance. Composable and highly reusable hardware units increase productivity. Hardware units use standardized interfaces for data exchange between them. Specifically, we chose a *stream interface* with flow control, appropriate for signal and image processing.

Our software design methodology builds applications using VSIPL++[1]. VSIPL++, a successor to the Vector Signal and Image Processing Library (VSIPL), addresses portability, performance, and productivity issues for embedded high-performance software design. The High Performance Embedded Computing Software Initiative (HPEC-SI)[2] is standardizing VSIPL++. VSIPL++ uses C++ object-oriented language features to improve the readability and expressiveness of programs, while delivering performance on par with traditional C and FORTRAN programs. Generic programming with templates enables custom compilations using machine features such as Single Instruction-Multiple Data (SIMD) instruction sets and the Portable Expression Template Engine[3].

Our infrastructure exchanges data between VSIPL++ and hardware streams, effectively combining hardware and software design. Communication across the software/hardware interface requires the conversion of data formats from VSIPL++ *blocks*, that is, memory-mapped data, to streaming data. We currently convert formats with *memory adapters*, hardware units on the FPGA that either read from FPGA memory to a flow-controlled stream of data or visa versa. `FPGABlocks`, a form of user-defined blocks storing data on the host side, behave in VSIPL++ just like built-in VSIPL++ blocks. A standard description of the functionality implemented on an FPGA helps manage possible FPGA configurations. This configuration file is read at run-time and `FunctionObjects`, corresponding to actual functions implemented on the FPGA, are created. Calling an `FPGAApply` routine moves data onto the FPGA from the hosts and configures the appropriate memory adapters to initiate the operation defined by a function object. `FPGABlocks` are *lazy*, i.e., they only move between the host and FPGA when necessary.

The methodology enables development of hardware to begin well before the target system is available. Individual units can be incrementally integrated and tested. Groups of units, sub-systems, and systems can be integrated into software and tested in the same way. Systems too large for a single FPGA can be spread across multiple FPGAs or multiple FPGA boards on network-connected machines. The stream interface makes this possible by allowing an arbitrary no-op to be placed between two units that will be

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
20 AUG 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
VSIPL++/FPGA Design Methodology	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Research Laboratory/IFTC**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**37**	

directly connected in the target system. In a virtual breadboard environment, this no-op transports a stream between multiple FPGAs, on the same board or on different network-connected hosts.

Our design methodology allows tightly coupled hardware-in-the-loop simulations to be easily constructed. Hardware-in-the-Loop Simulation is the practice of open or closed-loop simulation connecting software (often scene generation or sensor emulation) and hardware (often processing or guidance). The software portion of a hardware/software design implements the soft portions of the target system and simulates the model environment. The flow control nature of stream processing supports a mode of test where hardware operates at speed for a short period and then stalls while new stimulus is created/loaded (i.e. time stops).

We currently employ this methodology in two applications: the embedded system design of a space-based radar and the creation of a signal processing library for application acceleration. In a joint effort with the Jet Propulsion Laboratory, we are developing an on-board processor for a demonstration Moving Target Indicator/Synthetic Aperture Radar (MTI/SAR) space-based radar scheduled to fly in 2008. An FPGA front-end is capable of SAR processing (range compression, azimuth compression) and MTI preprocessing (pulse compression, Doppler filtering). A programmable backend performs MTI processing (adaptive Space-Time Adaptive Processing (STAP)). Although the system will not be fabricated until 2005 or flown until 2008, this design methodology permits hardware design to begin today, and a virtual breadboard of the entire system can be constructed on AFRL's hybrid cluster.

We are developing RStream, a set of stream components that implement common signal processing filtering, transforms, and utility functions. Some of these functions will be used for SBR, but the goal is to provide a library that can be used for VSIPL++ application acceleration.

At the time of this paper, we have implemented a prototype implementation of VSIPL++/FPGA integration extension for the Annapolis Firebird FPGA card, which uses Xilinx VirtexE FPGAs. We are currently extending this implementation to support the Annapolis Wildstar-II FPGA board, which contains two Xilinx Virtex-II 6M gate FPGAs. We are performing preliminary experiments on the Xilinx Virtex-II Pro, an FPGA which integrates a hybrid system (PowerPC processor and reconfigurable logic) onto a single chip.

There are a number of exciting areas for future work. As the benefits and limitations of our initial streaming protocol become better understood, opportunities arise to develop additional streaming protocols with complementary characteristics (e.g., blocks larger than a single word, embedded control). Currently we only interface between hardware and software with memory adapters. A number of other adapters are possible, such as a FIFO adapter that sends the data in a block to the FPGA for processing without logically placing the block onto the FPGA. There are a number of Computer Aided Design (CAD) future work areas. Finally, we could investigate other areas besides signal processing applications and embedded systems that could benefit from FPGA acceleration.

References

[1] VSIPL Vector Signal Image Processing Library http://www.vsipl.org

[2] HPEC-SI High Performance Embedded Computing Software Initiative http://www.hpec-si.org

[3] PETE Portable Extension Template Edward M. Rutledge, MIT Lincoln Laboratory
http://www.acl.lanl.gov/pete/html/index.html

ITT Industries
Engineered for life

VSIPL++ / FPGA Design Methodology

Capt. Jules Bergmann, AFRL/IFTC
Susan Emeny, ITT
Peter Bronowicz, ITT

Overview

- Introduction- Designing for Hybrid Architectures
- Design Methodology
- VSIPL++ / FPGA Integration
- Integration and Test
- Applications
- Status
- Future Work
- Conclusions

Introduction

- Hybrid Computer Architectures
  - FPGAs and Programmable Processors have the potential to deliver high performance
- Commercially Available
- Challenge of hybrid architectures to develop a methodology that will:
  - Exploit their capabilities effectively while
  - Making FPGAs accessible to a larger community of developers.

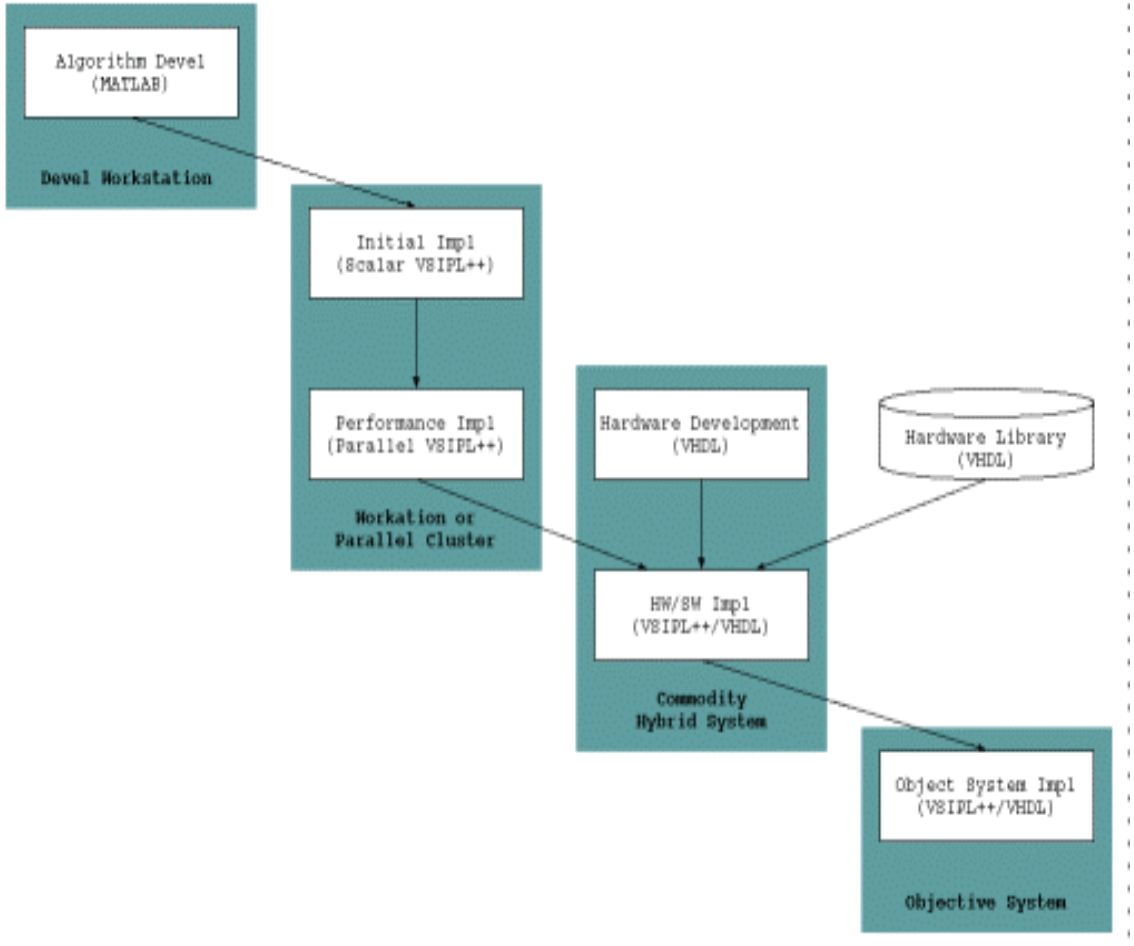
Requirements

Requirements of the Methodology

- Hardware & Software development needs to begin early
- Portable from test to final system, minimal change
- Scalable to future technologies, minimal change
- Productive
  - Concise function description for both HW & SW
  - Streamlined interface between HW & SW
- Use ***existing*** hardware and software methodologies

Design Methodology

- Algorithm
  - High level exploration (Matlab)
- Software
  - Scalar, C++, VSIPL++
  - Performance Imp. (parallel)
- Hardware
  - VHDL
  - FPGA Performance libraries
- Integration
  - HW/SW Debug on commodity cluster
  - Migrate to target system



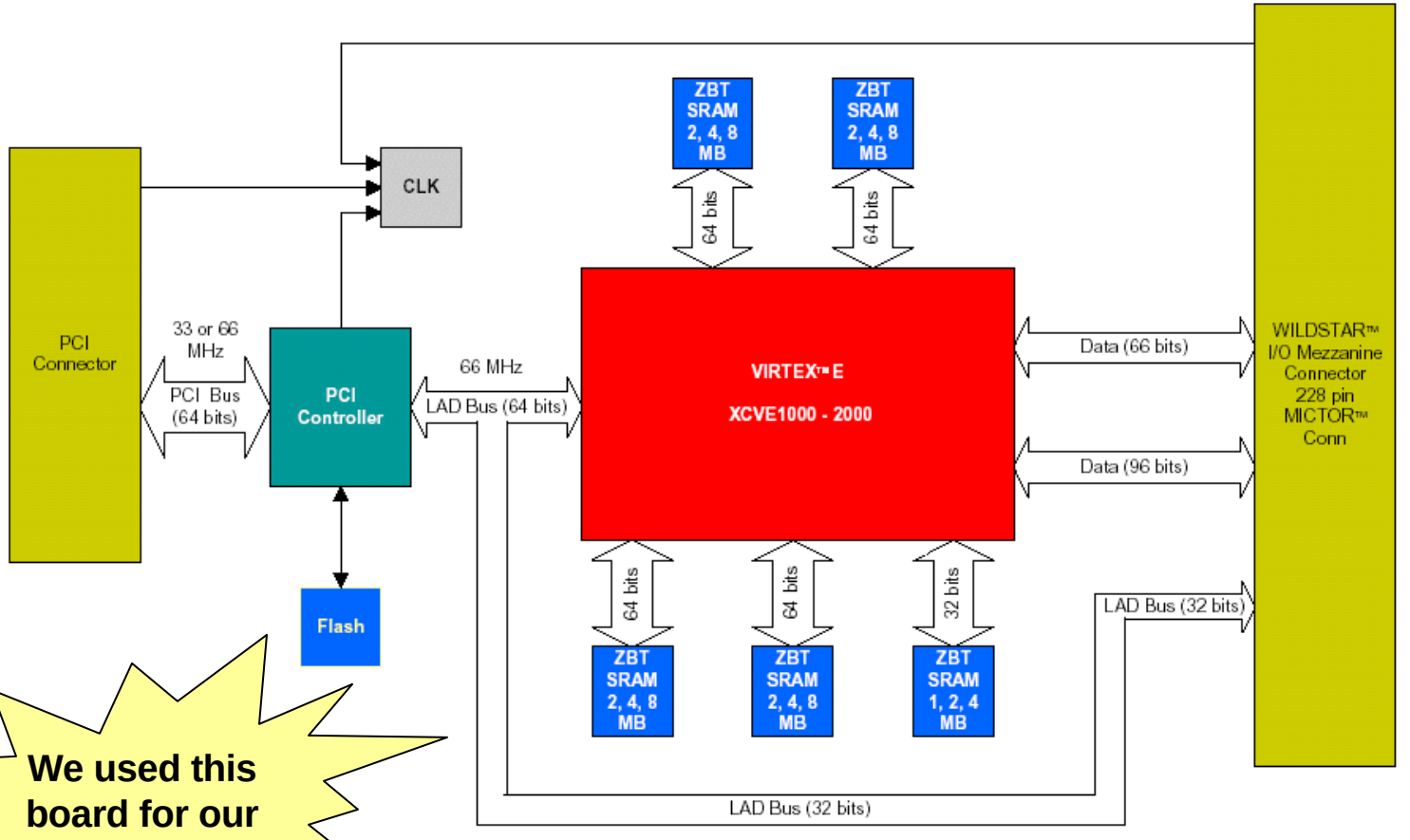
Benefits of the Methodology

- Support for System Design from algorithm to embedded system
- Simplifies integration of hardware and software
- Application acceleration via pre-defined FPGA libraries
  - Standard functions (fft)
  - Custom

Hardware Design Methodology

- Portability & Scalability
  - Use standard high level synthesis (RTL)
- Performance
  - Encapsulate device specific optimizations in macro cells
- Productivity
  - Standardize interface between units for data exchange
  - Use stream interface with flow control. This model matches the way data is usually produced from sensors and requires minimal assumptions about environment

Commodity Hardware



We used this board for our 1st Prototype!

Annapolis Firebird PCI Board

Software Design Methodology

VSIPL++ was chosen to achieve:

- Portability, the reference version compiles anywhere!
- Scalability, builds on existing standards i.e.. MPI
- Performance
  - Allows for optimized libraries which take advantage of specific machine features (transparent to application)
  - Allows for user defined functions (i.e. specialized or optimized FPGA functions)
- Productivity
  - Express computation in a natural fashion

ITT Industries
Engineered for life

Co-Design Issue

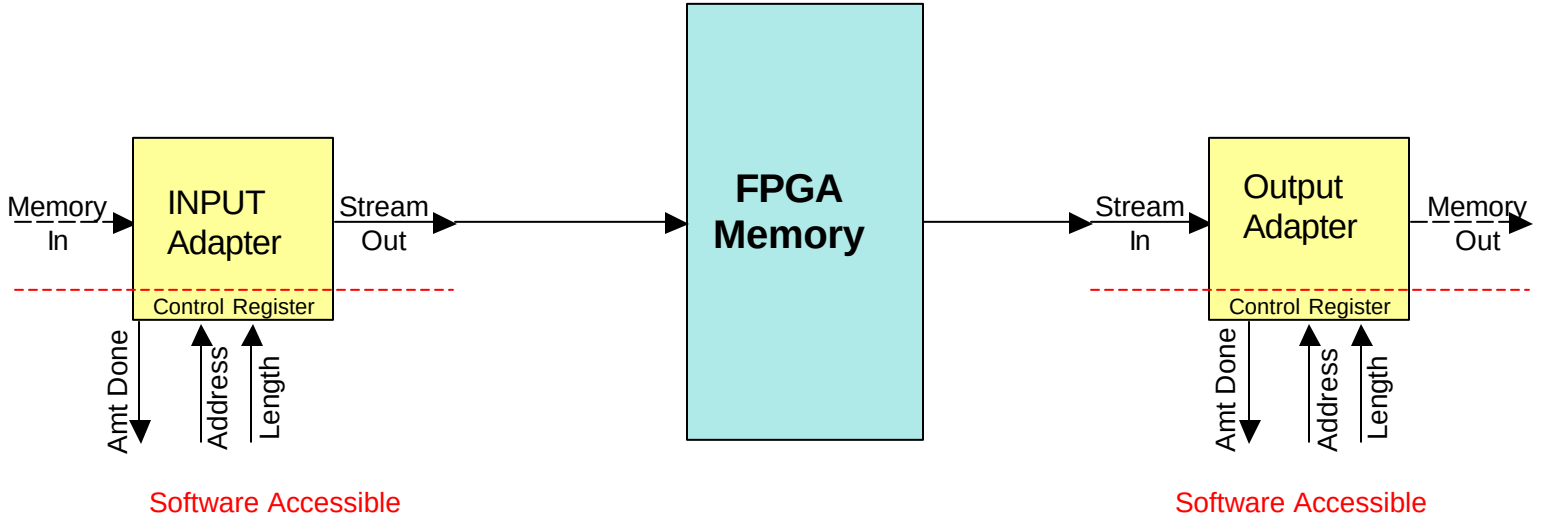
- VSIPL++ for software & Synthesizable compose-able modules for hardware are great domain specific methodologies.
- Software and Hardware treat data differently
  - VSIPL++ represents data in discrete *blocks*
  - Hardware sees the data as a continuous *stream*
- Poses a problem of data exchange
- We developed an infrastructure to exchange data between VSIPL++ block data and the FPGA streaming data

Co-Design Solution

- Devise a memory adapter, a hardware unit on the FPGA, that directly accesses FPGA memory
  - Uses DMA to place/retrieve data into/from a flow controlled data stream.

- Create a user defined VSIPL++ block (fpgaDense) based on the existing (Dense) block to facilitate the transfer of data to/from the FPGA memory

Memory Adapters



Control Data Flow

- The VSIPL++ FPGA Dense block is LAZY!
  - Data is only transferred when necessary. Vectors created on the host are copied to FPGA memory only when FPGA function is initiated.
  - Conversely data is written to host memory, only when the host requests the data

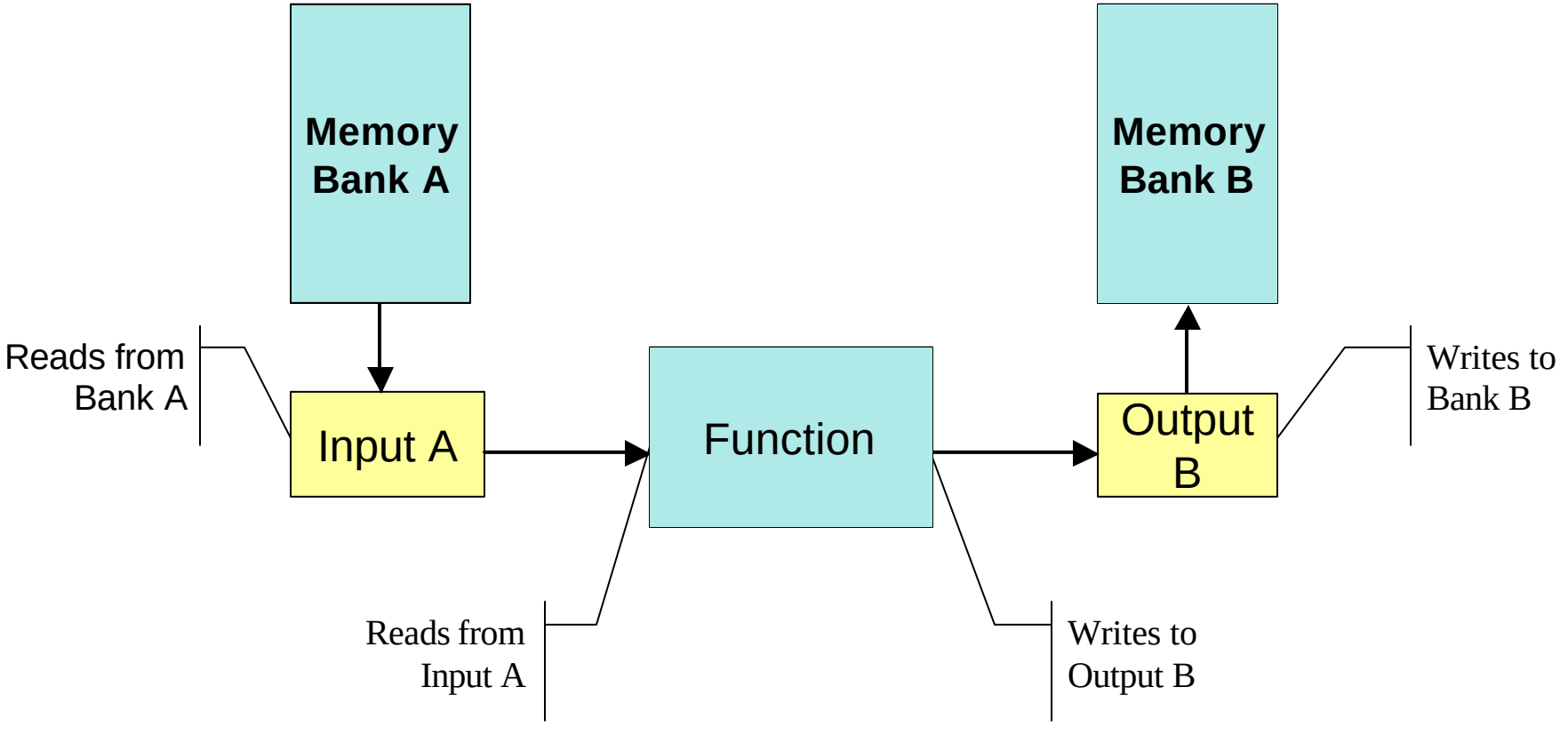
Software Representation

- The hardware is “Object Oriented” where each component object is described by connections.

- Therefore, we created software objects that directly corresponded to the hardware, also described by connections.

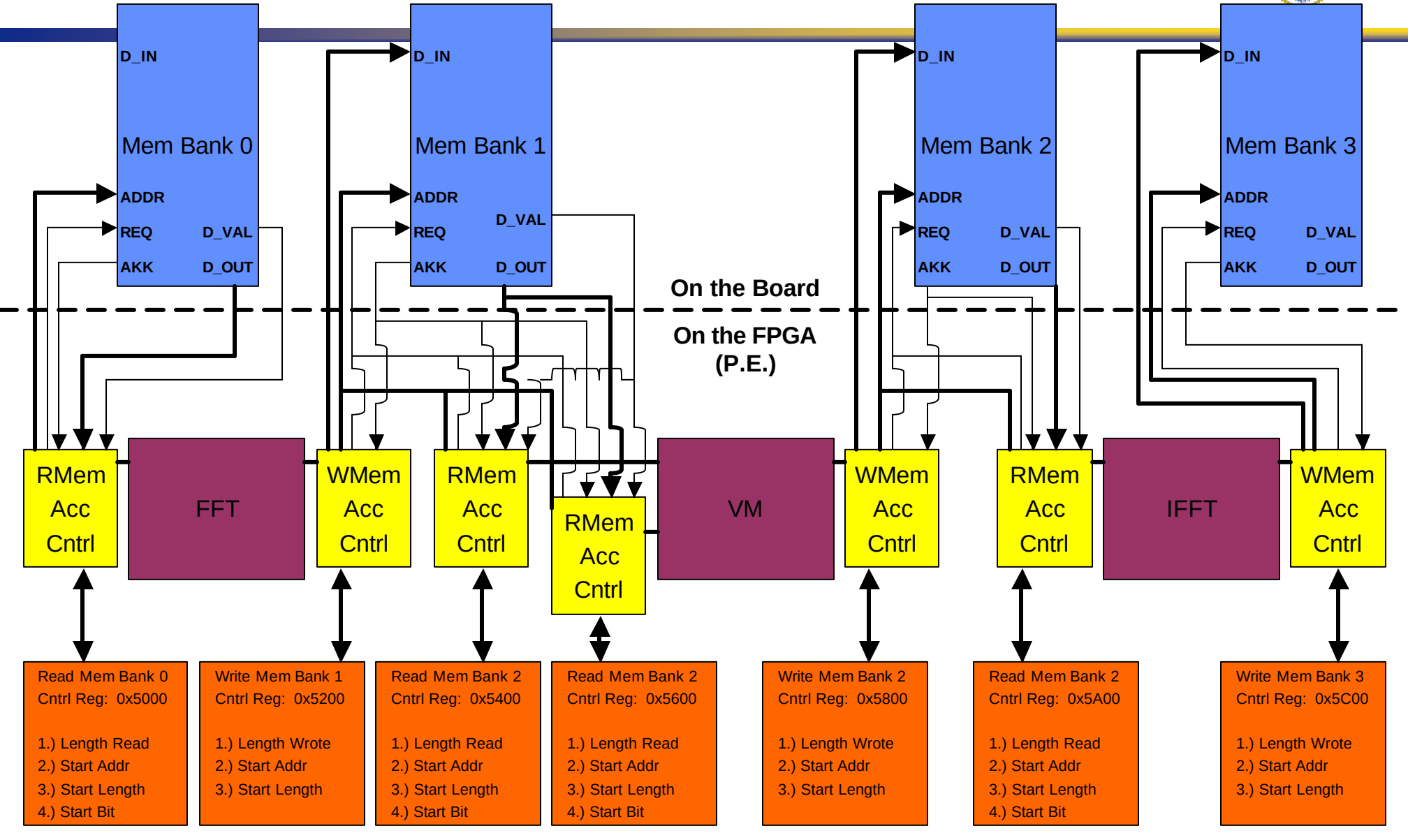
Facilitating the co-design of hardware and software.

Hardware Model

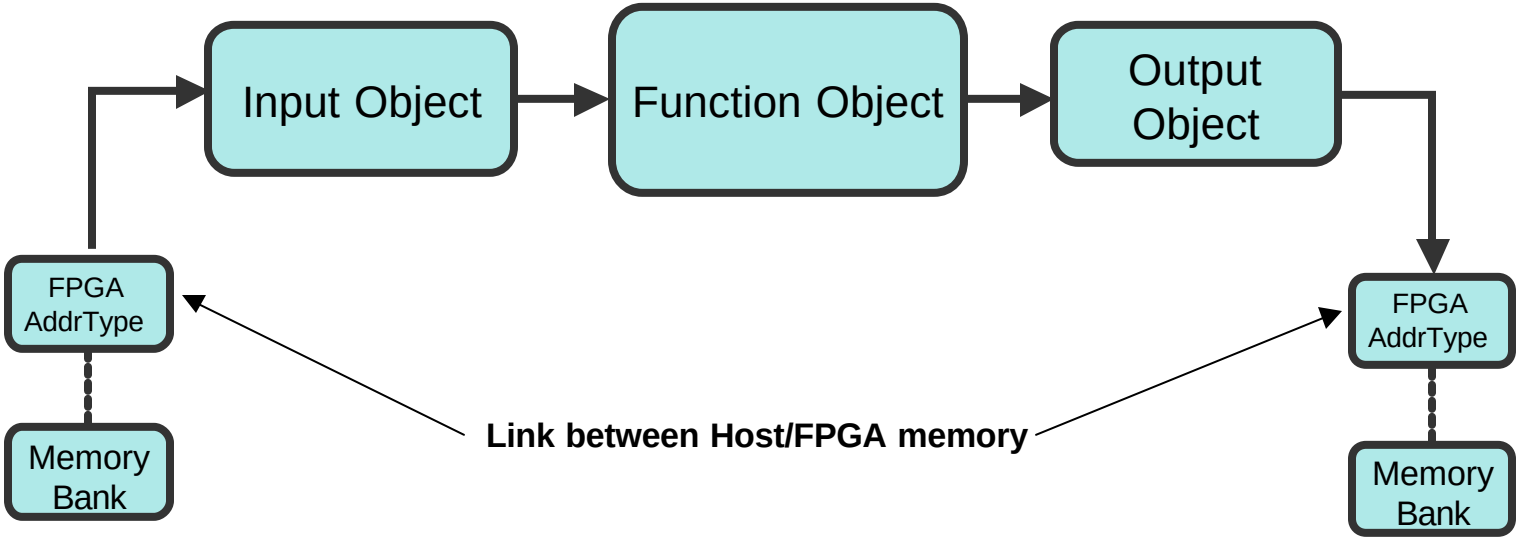


Arrows Represent Data Flow

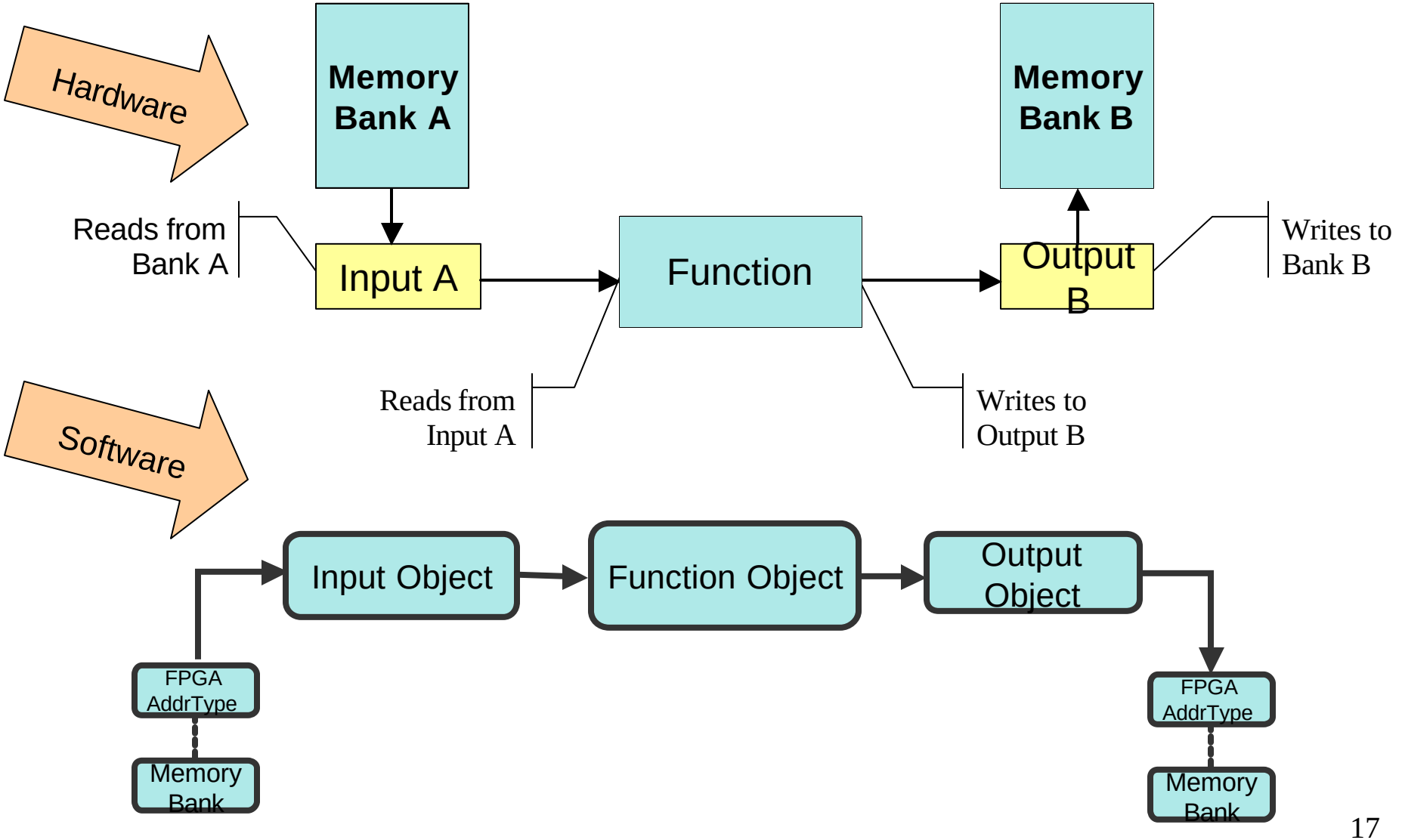
Hardware Block Diagram



Software Model



The Models


Hardware
Memory Bank A
Memory Bank B
Reads from Bank A
Input A
Function
Output B
Writes to Bank B
Reads from Input A
Writes to Output B
Software
Input Object
Function Object
Output Object
FPGA AddrType
Memory Bank
FPGA AddrType
Memory Bank

About the Models

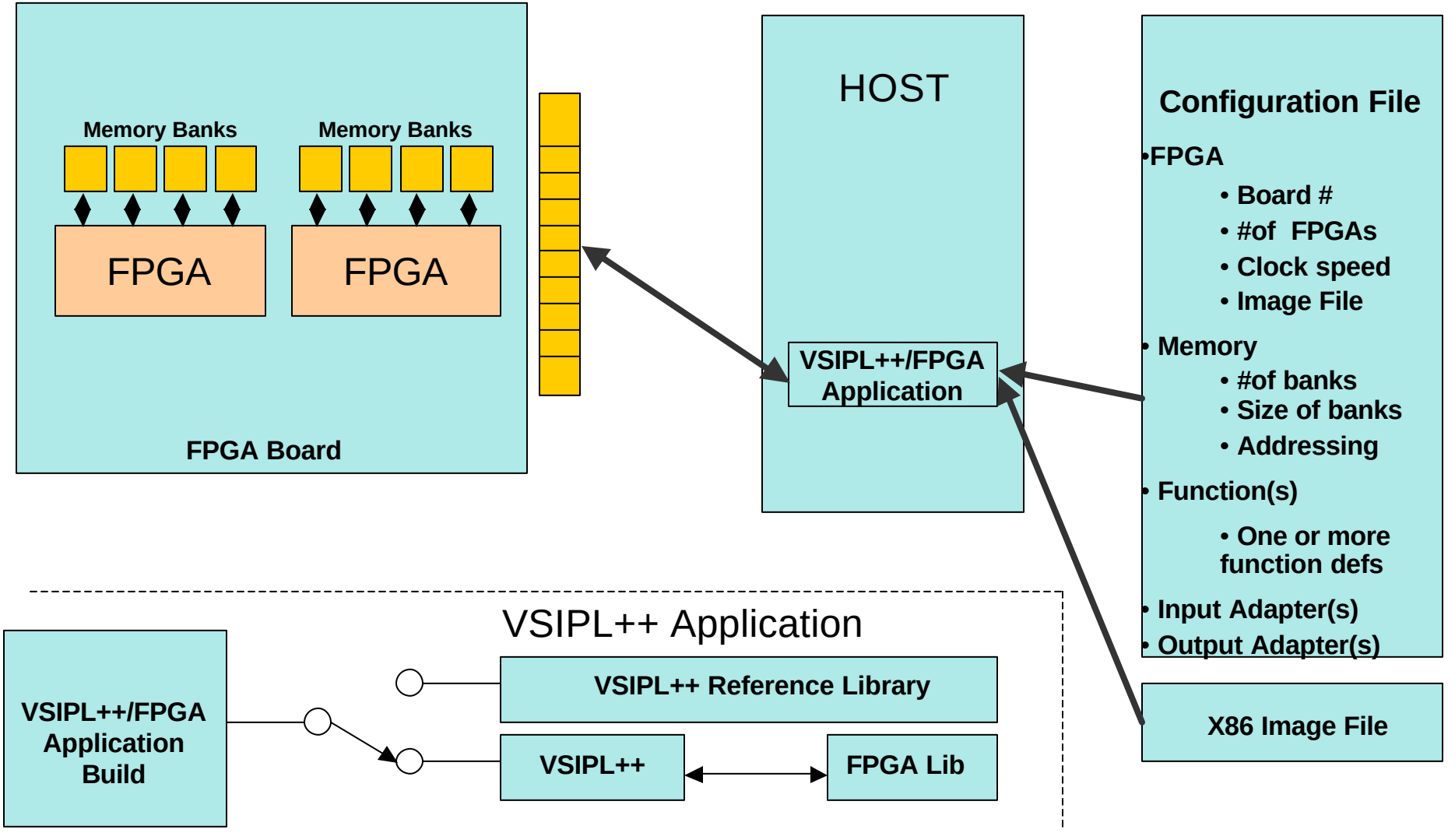
Both models, each object is described connections

- Functions are described by I/O connections (data flow)
- I/O is described by memory connections
  - Or another I/O (an output can feed an input)
- Memory is described by the board and the processing element it is connected to.

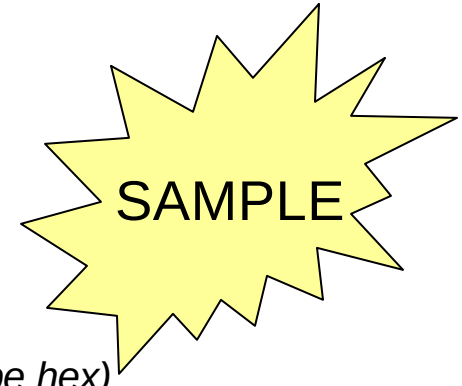
Integrating VSIPL++ & FPGA

- A description of each hardware object is contained in a Configuration file.
- The system software reads the configuration file and creates a software object for each hardware object.
- The application simply calls the “function”, the system will “apply” the function.
  - Move the data (if necessary) & initiate the operation.

VSIPL++ / FPGA Interface



Configuration File



#Configuration File

PE = 0, 20 *# Processing Element, processing speed mHz*

CoreFileName=pe0.x86 *# Filename of Binary code to program fpga with*

#Memory Bank Definition

#Name, PE#, Size in bytes, DMA Write Port, DMA Read Port, radix assumed to be hex)

MemBank=Bank0, 0, 400000, 1000, 1200 *# 4194304 bytes*

MemBank=Bank1, 0, 400000, 2000, 2200

MemBank=Bank2, 0, 400000, 3000, 3200

MemBank=Bank3, 0, 400000, 4000, 4200

#Function Definition

FN=PC, input=R0, input=R1, input=R2, input=R3, Output=W0, Output=W1, Output=W2, size=256

FN=FFT0,input=R0, Output=W0,Size=256

FN=VMUL, input=R2, input=R1, output=W1, size=256

FN=IFFT,input=R3, Output=W2,Size=256

Config File Cont.

#Input Adapter Defintion

#Name, Control Register Address, Name of assoc. memory bank, radix assumed to be hex

INPUT=R0, 5000, Bank0

INPUT=R1, 5400, Bank1

INPUT=R2, 5600, Bank1, persistant

INPUT=R3, 5A00, Bank2

#Output Adapter Definition

#Name, Control Register Address, Name of assoc. memory bank, radix assumed to be hex

Output=W0, 5200, Bank1

Output=W1, 5800, Bank2

Output=W2, 5C00, Bank3

Integration & Test

- Unit & System Level
  - As individual hardware units are developed, they can be immediately integrated into the software for testing, even before all components are complete.
  - Components can be integrated individually or grouped as sub-systems.
- Virtual Breadboarding
  - For larger systems, functions can be spread across multiple FPGAs.
- Hardware in the loop
  - Tightly coupled hardware/software can be easily constructed.

Applications

- Currently employing this technology in two applications:
  - Spaced Based Radar Embedded System design
    - With this method, a virtual bread board of the MTI/SAR system can be developed and tested on AFRL's hybrid cluster long before the actual hardware becomes available. (Not scheduled to fly until 2008).
  - *Rstream* signal processing library for application acceleration.
    - Develop a set of common Signal Processing functions which will be used in Space Based Radar.
    - Goal is to provide a library that can be used with VSIPL++

Status

- Prototype implementation completed
  - Annapolis Firebird Card using Xilinx VirtexE FPGAs
- Preliminary experiments on XILINX Virtex-II Pro which integrates a hybrid system, PowerPc processor

Future Work

- It is anticipated, that opportunities will arise as we gain experience with the streaming protocol
  - Possibly additional streaming objects (fifo's that logically bypass the fpga memory?)
- Computer Aided Design
  - Auto generate the configuration file
  - Auto merge of units that will be directly connected, optimizing redundant interfaces.
- Investigate areas outside of signal processing

Conclusions

- The integration of VSIPL++ for software design with compose-able hardware design, provides a powerful design methodology for building hybrid hardware/software systems.

- VSIPL++'s performance, portability and productivity provide a growth path for parallel performance and hardware acceleration.

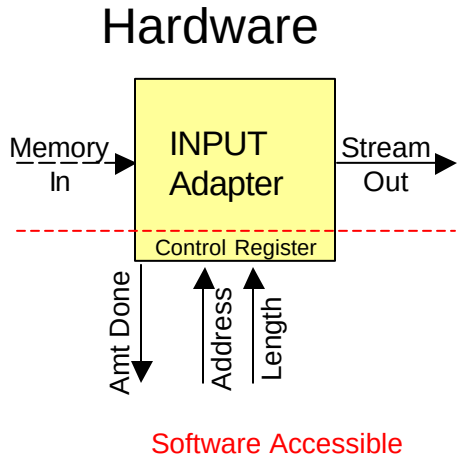
- The Stream hardware interface provides a simple mechanism by which hardware units can be composed together forming larger more complex units

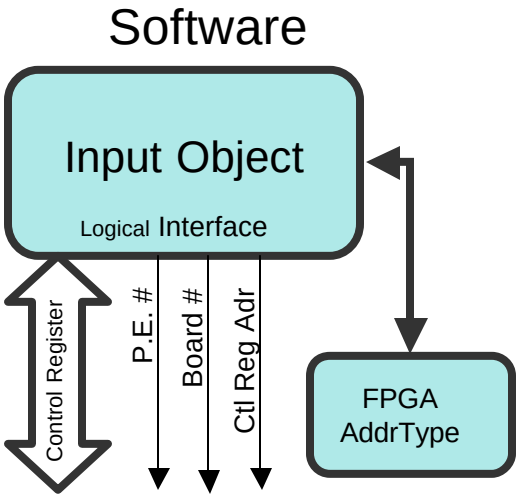
ITT Industries
Engineered for life

Addendum

Implementation Slides Follow

Input

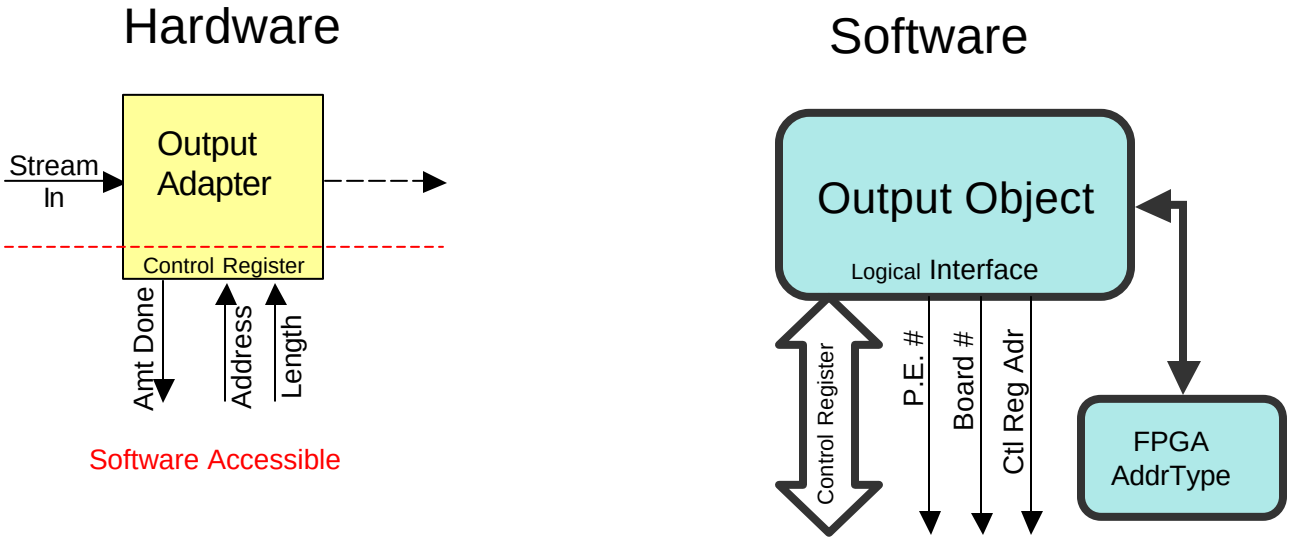




Input Responsibilities

- Control Data Flow
- Initiate Host memory to FPGA memory data transfers
- Initiate the data flow from FPGA memory to the function block

Output



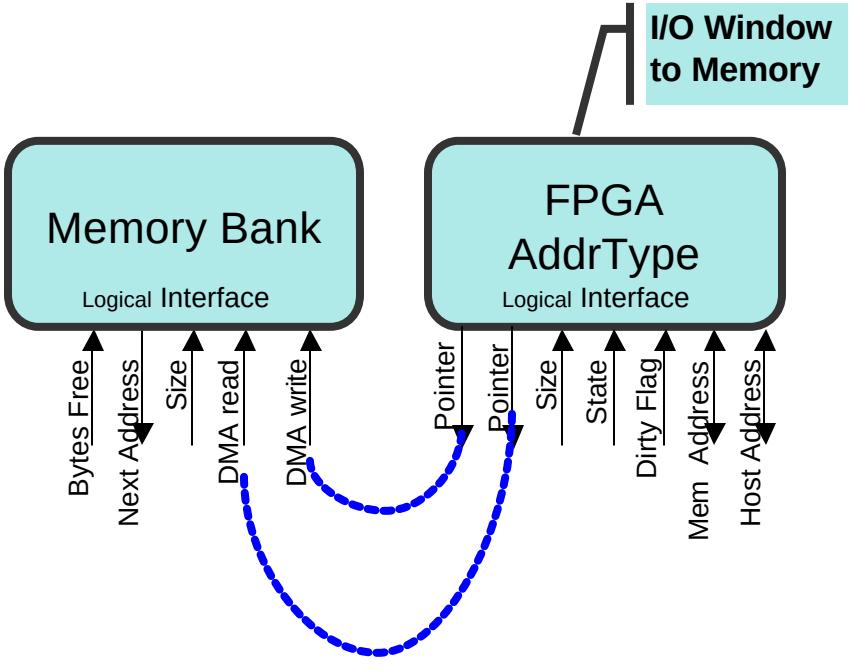
Output Responsibilities

- Control data flow
- Initiate FPGA memory to Host memory data transfers
- Initiate the data flow *to* FPGA memory *from* the function block

Memory Bank

Responsibilities

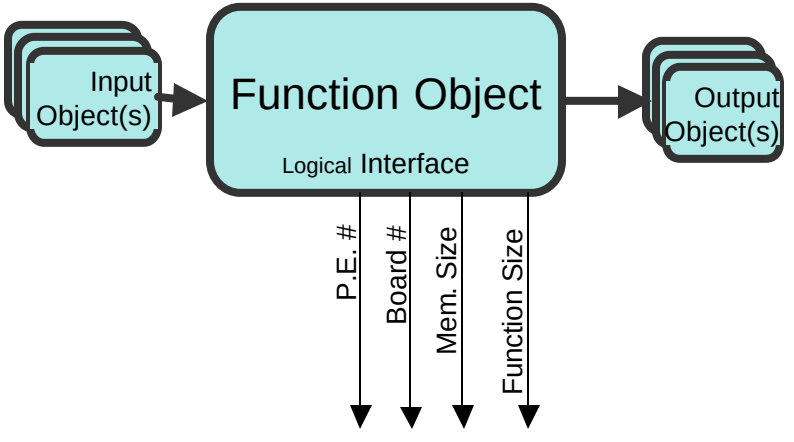
- Manage 1 FPGA memory bank
- Maintain Total size
- Maintains Available Free memory
- Honors requests for memory
  - allocate
  - free
- FPGA AddrType Object is I/O adapters “window” to the Memory Bank
  - Host/FPGA memory transfers

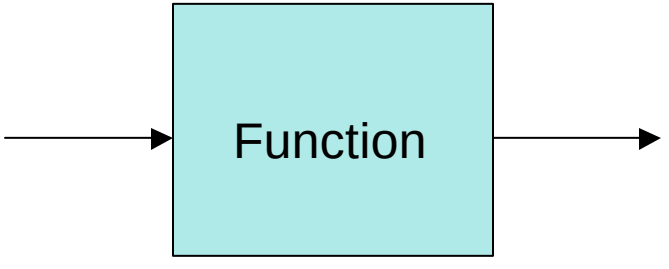


Function Object

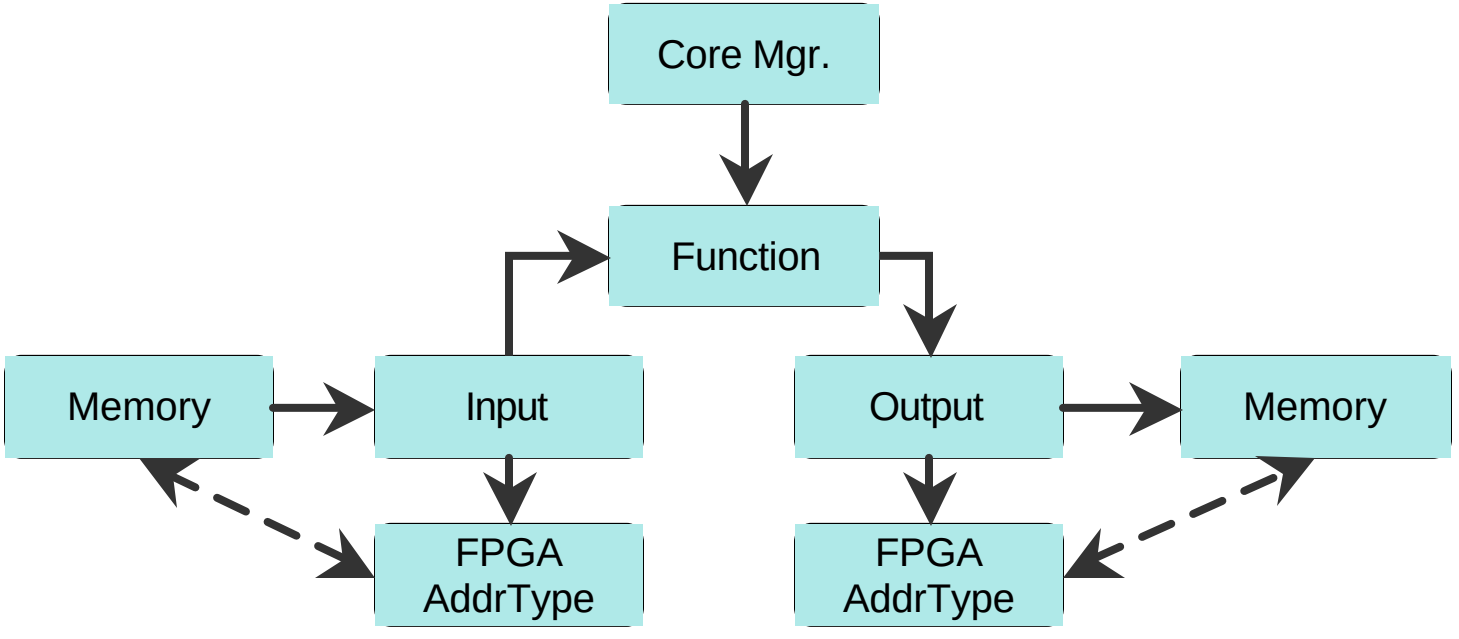
Responsibilities

- Instantiate Input Objects
- Instantiate Output Objects
- Activate/Halt Input Data Stream
- Activate/Halt Output Data Stream

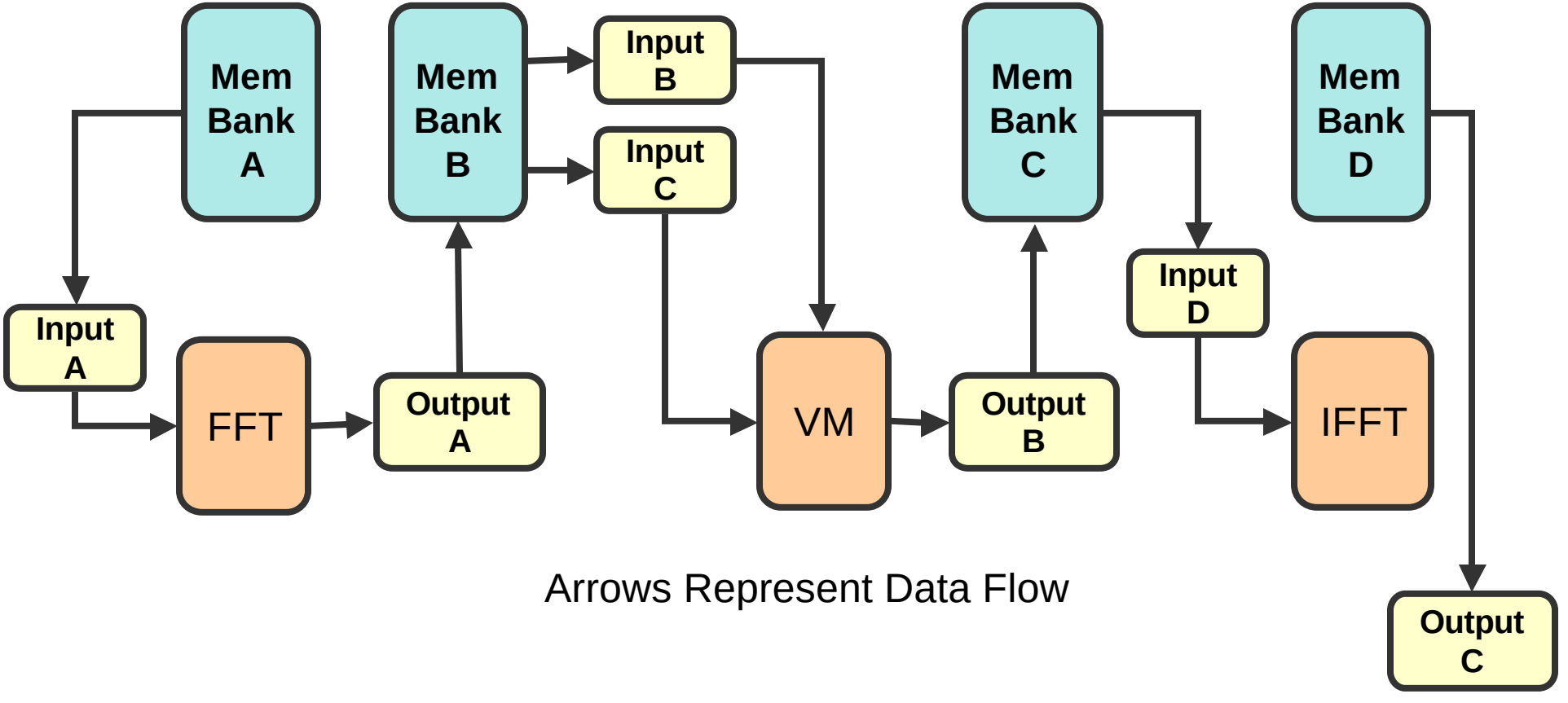




Object Relationships



Software Design

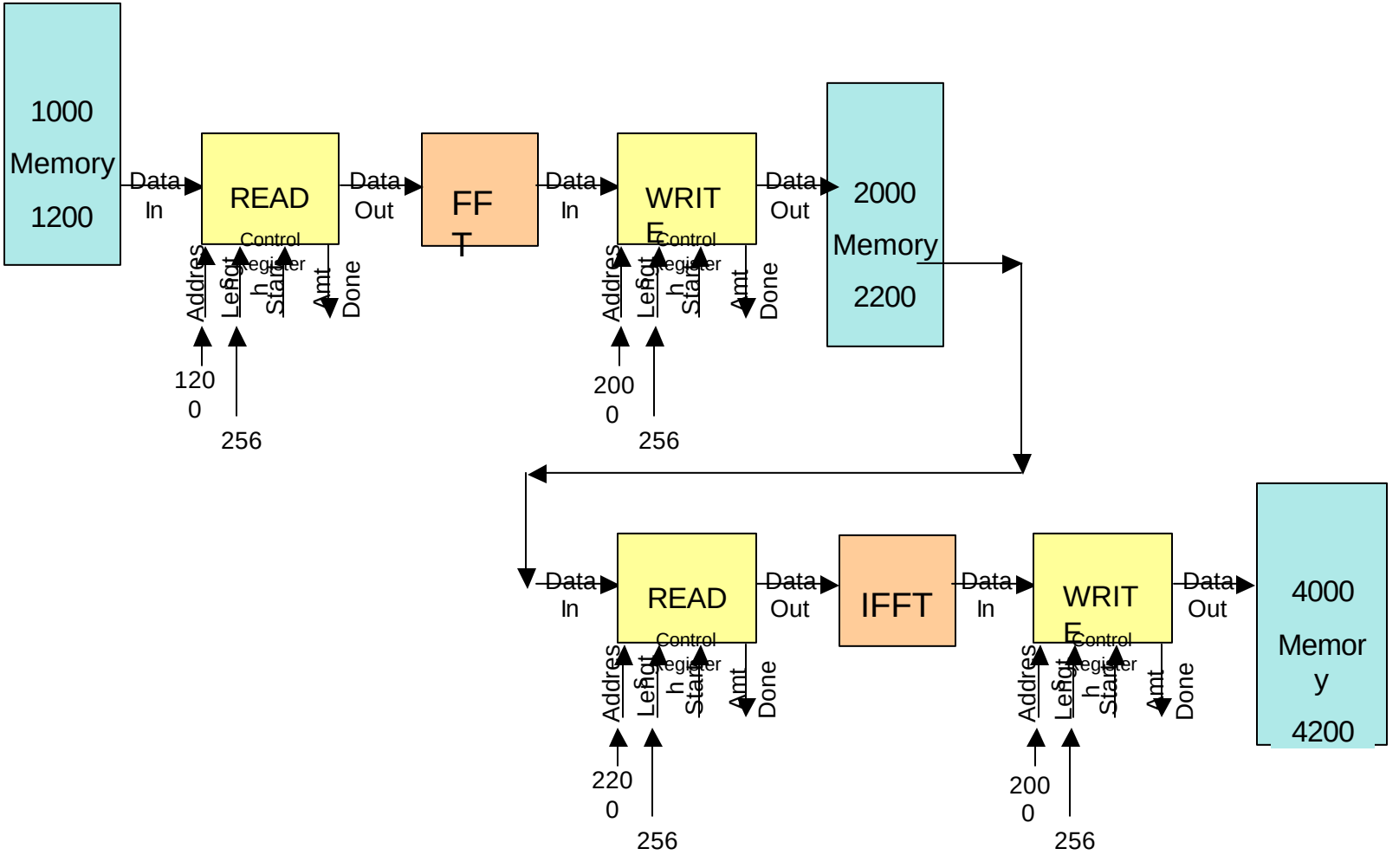


Arrows Represent Data Flow

PULSE COMPRESS

ITT Industries
Engineered for life

Hardware Design



PULSE COMPRESS